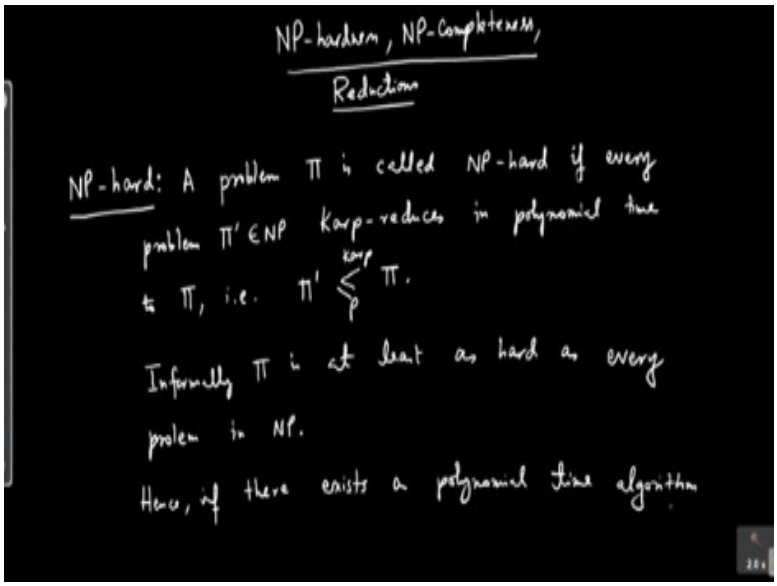
Selected Topics in Algorithm
Prof. Palash Dey
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 33
NP – Completeness of 3SAT

Welcome so in the last class we have discussed reductions namely Turing reduction and Karp reductions. Today we will see lots of examples of these reductions.

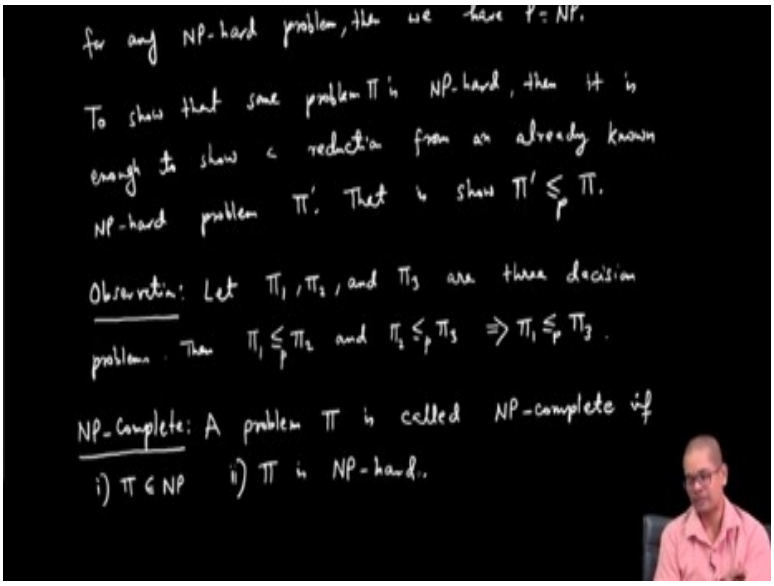
(Refer Slide Time: 00:38)



So, today's topic is NP complete reductions NP hardness, NP completeness and reductions. So, we have defined NP hardness informally that problems as hard as NP but now we define once now we have learned reductions. So, formally let us define NP hardness a problem Π is called NP hard if every problem Π' in NP reduces to or Karp reduces in polynomial time to Π that is Π' Karp reduces in polynomial time to Π.

And this is what do we mean by Π is at least as hard as Π'. Informally Π is at least as hard as every problem in NP hence. That is why the by the definition and hence if there exist a polynomial time algorithm for any NP not problem that means P = NP.

(Refer Slide Time: 04:20)



Hence if there exists polynomial time algorithm for any NP hard problem then we have P = NP all problems in NP can be solved in polynomial time. Now to show that some problem is NP hard then it is enough to show some problems called Π because in here it is enough to show a reduction from and already known NP hard problem Π'. So, whenever we say reduction without adding any adjective it always means Karp reduction it is the defective standard reduction that is followed in computer science.

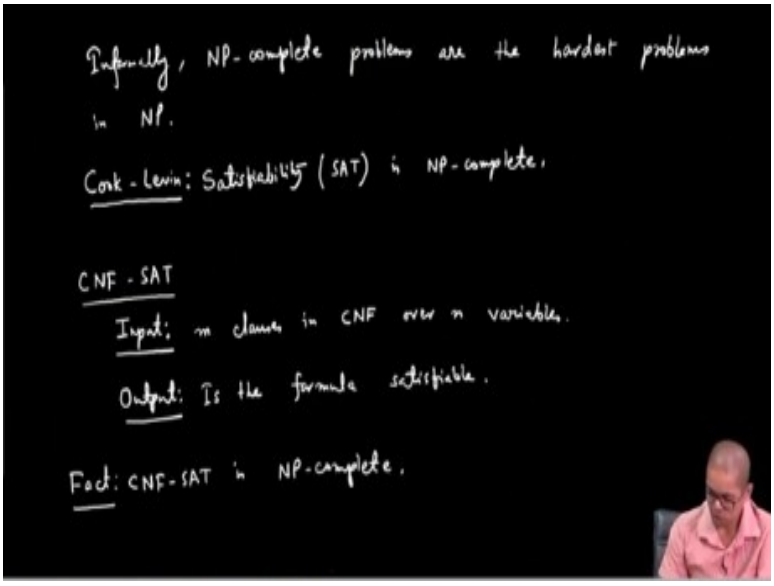
And there are some reasons, for example Turing reductions this complexity classes NP and co NP they are not closed on the Turing reduction. That means satisfiability Turing reduces to unsatisfiability but serviceability is a canonical NP problem and unsatisfiability is a canonical co NP problem. So, but this can this is not the case for Turing for Karp reductions you know the complexity classes NP and co NP they are closed under polynomial time Karp reductions.

So, to make the exposition easy we will drop the word Karp from the time being from now on. Now to show that you know some problem is NP hard all we need to show is that pick NP hard another NP hard problem say Π' and show a reduction from Π' that is show that Π' is at least as hard as Π and this works because of transitivity. So, here is another observation easy observation let Π_1, Π_2, Π_3 are three decision problems.

Then Π_1 polynomial time Karp reduces to Π_2 and Π_2 polynomial time Karp reduces to Π_3 implies Π_1 polynomial time Karp reduces to Π_3. So, hence if all problems can be reduced to pi prime that means and Π' reduces to pi hence all problem in NP reduces to Π and hence this establishes that this is Π is NP hard. Now what is NP completeness? NP complete a problem Π is called NP complete if Π belongs to NP that is one and second is Π is NP hard.

That means if there exists a polynomial team algorithm for Π then there exists a polynomial time algorithm for every problem in NP. So, in some sense NP complete problems are the hardest problems in the class NP.

(Refer Slide Time: 10:15)



So, so informally speaking NP complete problems are the hardest problems in NP and proving to prove some problem is NP complete all you need to show is that it belongs to NP that means if the instance then it can be verified by an efficient algorithm using a certificate using a polynomial time polynomial size certificate that is one. And the second is it is NP hard that means the there exists a reduction from one NP hard problem to this problem.

But this boils down to the fundamental question that how we prove the first problem to be NP hard. That means that we need to show from the definition that all problems in NP, reduces in polynomial time to that problem. So, using that definition we need to show the first problem and

that is the Cook Levin theorem they showed that all problems in P in NP polynomial time reduces to SAT satisfiability.

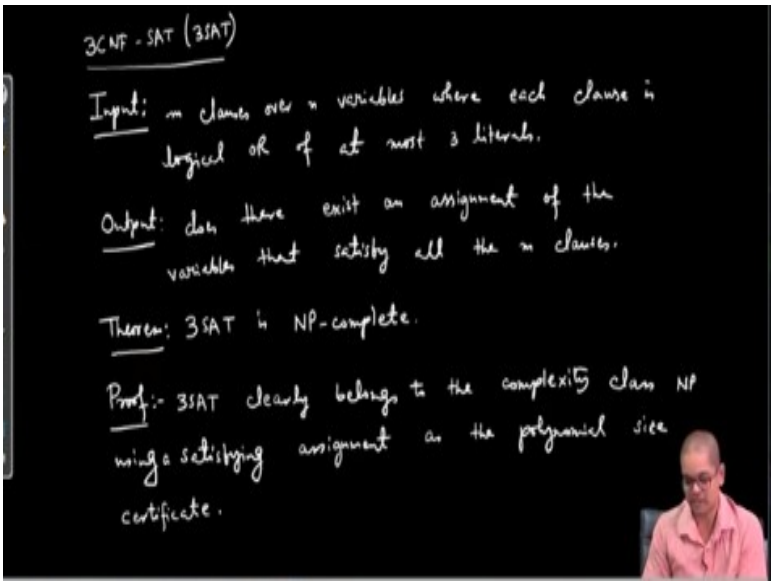
So, they showed that satisfiability also called SAT is NP complete who will not prove this the proof of this result is out of scope of this course and it is not in the it is not superbly important for our material but I would recommend all of you to look at this proof this is an interesting nice proof and not difficult. You need to understand the NP will and the definition and proof is intuitive enough. But we will not prove this.

So, what we will prove is that we will use this information that SAT is NP complete and using this we prove many NP complete many other NP complete problems. We see many other NP complete problems and we will see reductions. So, what is the SAT clause? What is the input of this of satisfiability? It is one possibility is arbitrary Boolean formula but another one is called CNF-SAT. So, CNF-SAT is also NP complete and in CNF-SAT.

We will also use the fact that CNF-SAT is NP complete and now from here on we will build up reductions and show that many other problems are NP complete so, input for CNF-SAT we are given input is m clauses in conjunctive normal form over in variables output again it should be a decision problem is the formula satisfiable. So, we will assume that this is known to be NP complete.

So, let me write it as a fact CNF-SAT is NP complete and using this, we prove that three CNF-SAT is also NP complete.

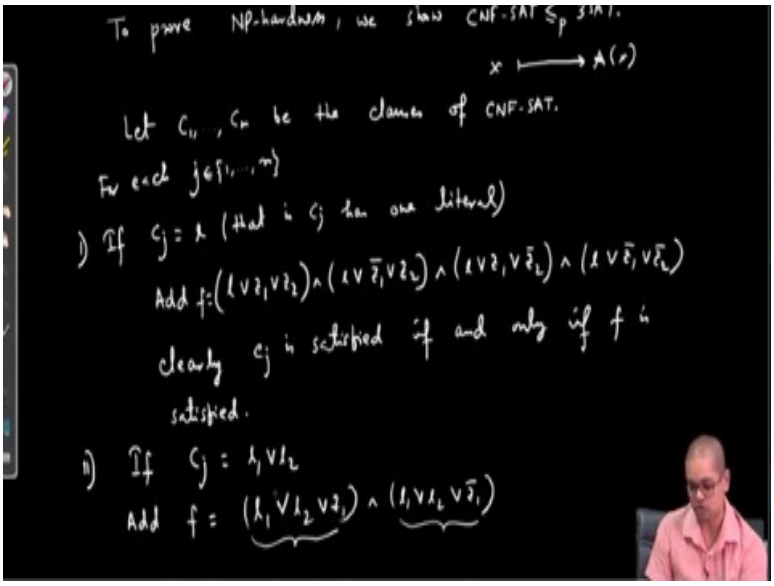
(Refer Slide Time: 15:30)



So, what is three CNF-SAT, is also abbreviated as 3SAT sometime is commonly abbreviated as 3SAT the input is m clauses over n variables where each clause is logical or of at most three literals and the output is question is does there exist an assignment of the variables that satisfy all the m clauses. So, what we will show now is that let me write it as theorem 3SAT is NP complete.

Proof, so by definition NP completeness has two parts one is membership in NP and another is NP hardness. So, does 3SAT belong to NP of course yes because and satisfying assignment could serve as a certificate. So, 3SAT clearly belongs to the complexity class NP using satisfying assignment using as the certificate as the polynomial size certificate.

(Refer Slide Time: 19:49)



Now to show NP hardness we exhibit a reduction from three CNF-SAT. To prove NP hardness, we show that 3 not 3 CNF-SAT polynomial time Karp reduces we are dropping this writing Karp every time polynomial time Karp reduces to 3SAT so, for that what I need to do? I need to convert an instance of CNF-SAT to an equivalence instance of 3SAT and that should be done in polynomial time in such a way that you know X is a YES instance X is satisfiable if and only if $A(x)$ is a YES instance that means $A(x)$ is satisfiable.

So, the only difference between CNF-SAT and 3SAT is that each CNF clause may not have may not involve three number of variables it can contain less number of variables or more than three variables. So, how to convert that? So, suppose so let $C_1, \ldots, C_m$ be the clauses of CNF-SAT now for each clause $j \in \{1, \ldots, m\}$ in a couple of cases if C_j is just involves one literal that is C_j has one literal.

So, we will convert each clause into a bunch of 3SAT clauses so that you know this clause C_j is satisfiable if and only if all of them are satisfiable, that is the idea. So, C_j if it is l then we introduce or add like this say add this clauses $(l \vee z_1 \vee z_2) \wedge (l \vee \bar{z_1} \vee z_2) \wedge (l \vee z_1 \vee \bar{z_2}) \wedge (l \vee \bar{z_1} \vee \bar{z_2})$. Let us call this if clearly C_j is satisfied if and only if f is satisfied.

So, let us verify this so if C_j is satisfied that means l is set to true then f is satisfied because l appears in each clause. So, f is satisfied. On the other hand, if f is satisfied, we claim that C_j

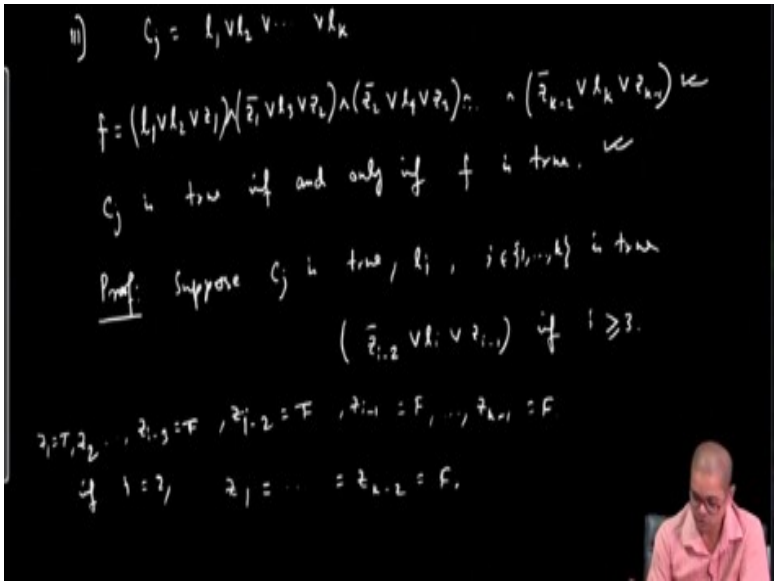
must be satisfied so suppose not. Suppose C_j is not satisfied that means l must be false. Now you ask how do z_1 and z_2 what are their setting what are the assignment now for each assignment at least one of these.

Both of them should be false and that particular clause will be false because l is also false. So, it follows that if C_j is false then f is also false. Among these four clauses exactly one of the clause will be false. Hence it is not satisfied. The next one is if C_j involves two variables two literals and this temporary variable z_1, z_2, they appear nowhere else they appear only in these four clauses.

So, if C_j is $l_1 \vee l_2$ what we do is that add $f = (l_1 \vee l_2 \vee z_1) \wedge (l_1 \vee l_2 \vee \bar{z}_1)$. Now here again if C_j is satisfied if and only if f is satisfied. Because its C_j is so let us see if C_j is satisfied then $l_1 \vee l_2$ turns evaluates to true if $l_1 \vee l_2$ relates to true then both the clauses of f evaluates to true. On the other hand, if $l_1 \vee l_2$ is false that means if C_j is false then if z is false then the first clause turns out to be false and if z is true z_1 is true.

The second clause turns out to be evaluates to false. Hence at least one of the clauses becomes false and hence f turns out to be false.

(Refer Slide Time: 26:58)



If C_j involves three literals, we do not have need to do anything because in 3SAT we are supposed to have three literals so just take that literal and put it in the 3SAT instance. So, suppose C_j involves more literals C_j involves say k literals $l_1 \vee l_2 \vee ... \vee l_k$. So, what we do is that we introduce a clauses like this $l_1 \vee l_2$ and no z_1 next and $(\bar{z}_1 \vee l_3 \vee z_2) \wedge (\bar{z}_2 \vee l_4 \vee z_3)$ and so on this continues till the last one that $z_{k-2}^{-} \vee l_k \vee z_{k-1}$.

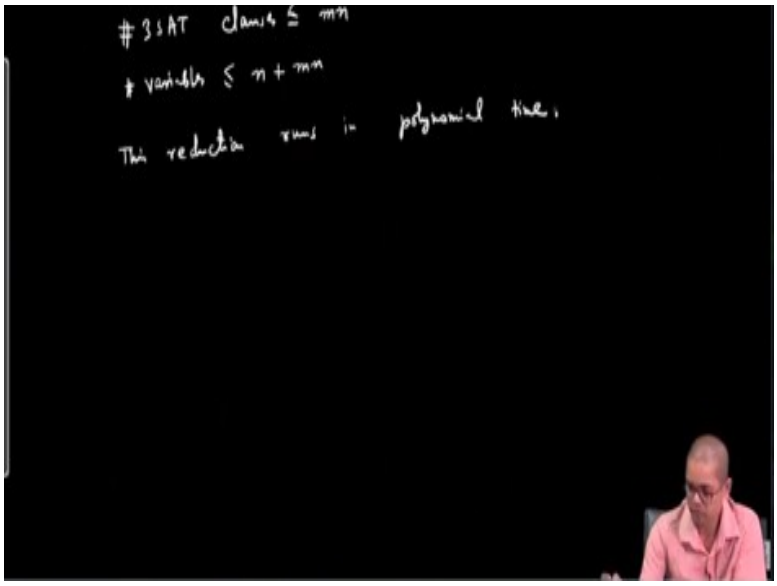
Now again here also we can check that you know C_j is true for any assignment of the variables if and only if f is true. Now here these things are a little non trivial to verify so suppose C_j is true so let us prove this so suppose C_j is true then one of these literals will be true suppose you know for if l_1 is true so if it is true and you know suppose $l_i, i \in [k]$, l_i is true. Then look at where does l_i is in l_i it looks like this if i is greater than greater than equal to 3.

Then this is looks like i - 2 or z_{i-1} that is how it looks like if i is greater than equal to 3. Then you set z_{i-2} to be false and z_{i-1} to be false and percolate this way that means you know z_{i-3} to be means all these are false up to this is z_1. So, z_1 is true and from z_2 to this much is false it is like and here sorry if l_i true then z_{i-2} to be true this is true from z_1 to z_{i-2} make it true and from z_{i-1} to z_{k-1} set it false.

On the other hand, if i is 2, 1 or 2 then set z_1 if i is 2 then set z_1 everything to be false z_1 to z_{k-2} to be false. Similarly in the other way suppose f is true A folds true then you see that you know all of them all z_1 to z_m no if all of them are false then either $l_1 \vee l_2$ is true and that means C_j is true. On the other hand, if there exist at least one of them true so what is the first one which is true from z_1 you will search from z_1 and see which one is the first one to be true.

And if z_{i-2} to be the last one to be true which is the last one to be true if z_{i-2} is the last one to be true then l_i must be true.

(Refer Slide Time: 32:37)



So, we have this that f is true if and only if z_{i-2} is true and does this run in polynomial time. So, the very number of clauses number of 3SAT clauses how many. You know each one we for each clause we involve at most the number of neutrals involved in that clause. So, the number of clauses is at most there are m clauses and each clause can have at most n literals so number of clauses in 3SAT instances at most m times n.

And number of variables is also the original n variables were there and, in each clause, we are involving at most number of literals involved minus 2 clauses. That means at most the number of literals clauses and the number of literals is at most n and there are m clauses so we are involving at most you know m times n clauses a m times n variables. So, this reduction runs in polynomial time.

So, this concludes the proof. So, in the next class we will see some more examples of reductions. Thank you.